An Open Platform for Teaching and Project Based Work at the Undergraduate and Postgraduate Level

Benjamin N. Passow, James Wheeler, Simon Coupland, and Mario A. Gongora

Abstract
Robots are a great tool for engaging and enthusing students when studying a range of topics. De Montfort University offers a wide range of courses from University access courses to Doctoral training. We use robots as tools to teach technical concepts across this wide and diverse range of learners. We have had great success using the Lego RCX and now NXT on the less demanding courses, and conversely with the MobileRobots Pioneer range for postgraduate and research projects. Although there is a distinct area in between these two where both these platforms meet our needs, neither is suitable for every aspect of our work. For this reason we have developed our own hardware and software platform to fulfil all of our needs. This paper describes the hardware platform and accompanying software and looks at two applications which made use of this system.
Our platform presents a low-cost system that enables students to learn about electronics, embedded systems, communication, bus systems, high and low level programming, robot architectures, and control algorithms, all in individual stages using the same familiar hardware and software.

Keywords: Teaching, Project Based Work, Undergraduate, Postgraduate, Robotics, Embedded Systems, Programming, Algorithms, Hardware, Software, Case Study

Introduction

Robots and control systems have become essential parts of modern industry and are increasingly used in education. Within many teaching curricula, pupils are often introduced to robots at the primary school stage, where they learn concepts such as direction, angles, measurement and sequencing. At this level, Roamers [1], Pixies [2], and BeeBots [3] are popular choices due to their simple programming interface and "friendly" appearance.

At a higher level, students may make use of their theoretical knowledge by applying these to a real world machine [4]. General computer science as well as robotics and artificial intelligence students begin to explore the mechanics of robot design, constructing their own robots and adding sensors and actuators to suit a particular challenge. In this format there is generally some form of processor unit or brain that contains the control instructions and connects to the sensors and actuators. The control software is often developed on a standard PC and then uploaded to the controller via a communications link. Common choices for this format are the Lego Mindstorms [5] RCX and NXT and the Robix Rascal [6]. This practice showed to be effective for motivating students in practical activities [7].

For teaching software processes relating to control systems, it is often desirable to employ a robot platform with standard actuators and sensors (e.g. having motion, vision, hearing, proximity detection etc) with an embedded PC as the central control processor. In this environment, students learn to write control software that uses the underlying operating system to communicate with the available sensors/actuators. Examples of such robot platforms include the MobileRobots Pioneer and Peoplebot [8].

At De Montfort University, whilst we have found the Lego Mindstorms kits and the MobileRobots equipment to offer extremely useful platforms for the various teaching courses offered, there are some concepts, such as electronic design and embedded programming, that neither platform allows us to teach in the way we would like. For this reason we have developed our own printed circuit board (PCB) with an onboard Microchip microcontroller and several I/O connections that easily interface to commonly used actuators and sensors. Since a student version of the Microchip Integrated Development Environment (including editor, compiler, debugger and programmer) is freely available, we may use this as the main environment within which students develop their embedded code. The Microchip In-Circuit Debugging tools are relatively cheap and provide a useful means for interfacing between a host PC and the robot control platform.

By using a modular approach to the design of the platform along with its accompanying electronic interfacing and software libraries, we are able to easily reconfigure the platform according to the nature of the concepts being taught. As an example, for first year students we can provide them with pre-built sensor circuitry and a software library of high-level C functions that enable them to design a simple embedded system whilst shielding them from the lower-level complexities of electronics and software. As the teaching programme progresses, the control platform can be reconfigured so that students are required to design their own electronic interfaces or write their own low-level software in order to accomplish the tasks assigned to them.

This paper describes the development of the platform and software libraries in more detail. We include two case studies highlighting how the platform has contributed to the

teaching programme at both first year Bachelor course level and also at Master and Doctoral training levels. Finally we offer a conclusion that summarises how this approach may be of benefit to other educational establishments with a robotics teaching programme.

1. The Platform

The hardware side of the platform consists of a printed circuit board with voltage regulation, a 16bit Microchip programmable interrupt controller (PIC), analogue and digital peripheral input and output pins, two RS232 serial ports, I^2C bus, pulse width modulation (PWM) and motor control outputs. The PIC is programmed via a commercially available USB in-circuit debugger. This section will introduce and discuss the platform in more detail.

1.1 *Hardware and its Components*

The design of the board is optimised for mechatronics and control projects. It is based around a Microchip microcontroller dsPIC30F4011, which can run at up to 30 million instructions per second (MIPS), has 48kB program memory, 2kB random access memory, 1kB non-volatile EEPROM memory and 31 I/O ports. The PIC is powered by 5 VDC for the digital power supply, which is regulated by a standard analogue voltage regulator LM7805. We used the TO-92 package to maximise the power dissipation capability so that a range of battery voltages can be used, up to an online-charging lead acid battery at 14.8VDC. Since this board is intended for robotics projects, it is assumed that it will be used with batteries only, not a mains power supply; as such it has no rectifier diodes or large ripple-filtering capacitors at the input. It includes only the compact capacitors required to filter the feedback and noise from the digital clock and circuitry and the power devices that might be connected (e.g. electric motors).

This particular PIC provides three PWM-specific outputs (balanced pairs of digital outputs); two of which are connected to a dual motor driver chip L298N. This provides two full H-bridge PWM direct motor power outputs from the PCB. The H-bridge driver chip provides an interface between the digital supply voltage (typically +5VDC) and the battery voltage (typically +12VDC), which supplies power directly to the motors through the H-bridge. We have tested powering motors from 7-12 volts from different types of batteries (e.g. 7.2V or 9.6V from an array of NiMH, 7.4V from an array of Li-Po and 12V from standard sealed Lead acid), and our system has shown to be quite effective for most applications. All standard protections are included in the PCB so that the students need only connect the motors directly; there is a set of flyback fast switching inverse diodes to ground and power VCC (battery) and capacitor in parallel with the motor. The third PWM set of outputs from the PIC is available for expansions in the projects via a connector in the PCB.

Four of the PIC's signals are dedicated for driving RC-hobbyist servos (pulse position controlled position-servo mechanisms). These position-servos draw the power from the 5VDC regulated power supply to avoid problems when using batteries above 9V, which would be outside the tolerance of such devices (typically designed to work between 4.8V - 9.0V). The outputs from the PIC are connected to four 3-pin headers arranged in the standard Ground-Power-Signal configurations used by most RC-hobbyist servos.

Finally, there are two more dedicated headers, both intended for communications. One uses one of the PIC's UART pins to connect to a standard RS232 serial port. The pins come directly to the headers so that the digital signals from the PIC are available directly, i.e. there is no RS232 level-converter driver on the PCB. This allows connecting directly to other digital serial ports. If a standard serial port is going to be used (e.g. to connect to a computer) then an external RS232 level converter (e.g. MAX232) is required. We have various mini-PCBs with a MAX232 already mounted for use in various projects. The other communications header provides digital signal connection to the I^2C port from the PIC. This is mainly used for connecting to peripherals such as ultrasonic rangers, electronic compasses or IMUs. The addressable structure of this serial bus allows multiple devices to be connected and it has proved to be very useful and versatile as there is a vast range of peripherals, sensors, etc. that are available commercially and at low cost using this protocol.

Quantity	Interface name and description
Actuators:	
2	Full H-bridge motor drivers
1	Full-balanced PWM digital output
4	Direct connections to PPM postion-servos
16	Simple digital actuators via I/O ports
Sensors:	
<127	I^2C sensors Available to our students are: · Digital Compass · Ultrasonic ranger · Other boards
9	Analogue sensors (1Msps @ 10bit) Available to our students are: · Light dependent resistor · Inertial measurement unit (IMU)
16	Digital sensors (various)
Communication:	
2	UART serial ports (RS232 via converter)
1	I^2C bus (master or slave mode)
Expansion:	
17	Additional programmable I/O pins

Tab.1 Platform Interfaces and available equipment

The remaining I/O pins of the PIC are connected to a general-purpose header, which the students can use to connect any other type of peripheral or device not covered by the other headers mentioned above.

This convenient and compact design provides the optimal configuration for robotic and control projects. Table I summarises the platform's available interfaces for the students to use.

1.2 *Development Environment and Tools*

The microcontroller is programmed and can be debugged using Microchip's in-circuit debugger ICD2. This device is connected via USB to the host machine running the integrated development environment called MPLab. The standard programming language that comes with this development environment is assembler. In order to program with a high level programming language, an additional cross-compiler is required. We use Microchip's C30 compiler which is freely available for research and student projects. The compiler is fully ANSI compliant and includes a set of libraries for easier device configuration and use.

2. Software Libraries

To enable students new to programming and robotics to work with the platform we have written a set of high level functions for them to use. This section details some of the software libraries that provide simple software interfaces to functionality such as timers, sensors, communication, and motor control.

2.1 Timers

At the heart of any embedded controller is a timing system, our system is no different. Our application programmable interface (API) supplies four basic functions which can be combined to give all timing functions necessary:

```
// Initialise timer device

void timePassed_init (void );

// Reset timer device

void timePassed_reset (void );

// Get elapsed time (ms) as a uint

unsigned int timePassed_ms ( unsigned char );

// Get elapsed time (s) as a uint

float timePassed_fs ( unsigned char );
```

The function timePassed_init sets up the timer by setting the relevant configuration bits on the PIC's timers. This function must be called before the other timing code will work. The function timePassed_ms returns the elapsed time in milliseconds as an integer whereas timePassed_fs returns the elapsed time in seconds as a floating point number. Elapsed time in both these functions is a measure of how much time (measured using processor clock cycles) has elapsed since the PIC timer was reset. The PIC timer is reset by four possible actions:

- Calling timePassed_init().
- Calling timePassed_reset().
- Calling timePassed_ms(1).
- Calling timePassed_fs(1).

Although the initialisation function must reset the timer, we also provide the explicit timePassed_reset() reset function. Additionally the timer may be reset when measuring the elapsed time by calling the relevant function with a parameter of 1. These functions provide a simple interface for measuring time in milliseconds and seconds.

2.2 Analogue to Digital Converter

The analogue to digital converter (ADC) provides access to readings from analogue sensors connected to our system. Our API provides four functions for controlling and accessing the sensor readings from the ADC:

```
// Initialise ADC

void myadc_init (void );

// Start the ADC reading timer

void myadc_startReadings (void );

// Stop the ADC reading timer

void myadc_stopReadings (void );

// Read data from the ADC

int sensorReading (char sensorNumber );
```

The ADC needs to be initialised, this is done by calling myadc_init(void). The initialisation routine sets up a timer driven interrupt system which reads data off the ADC according to a timer which can be controlled through the API. The timer is started and stopped using the myadc_startReadings(void) and myadc_stopReadings(void) functions. When the timer elapses it causes an interrupt routine to run with regular frequency. The interrupt reads data from the ADC to a predefined data structure via a mean of two filter. This data can be accessed through the sensorReading(char sensorNumber) function. This is in effect an interrupt-driven polling system – the ADC is polled with a regular frequency as designated by a timer. It is worth noting that the polling timer causes interrupts to be raised, meaning that although the ADC-API uses a polling system this could be modified to a pure interrupt driven system fairly easily.

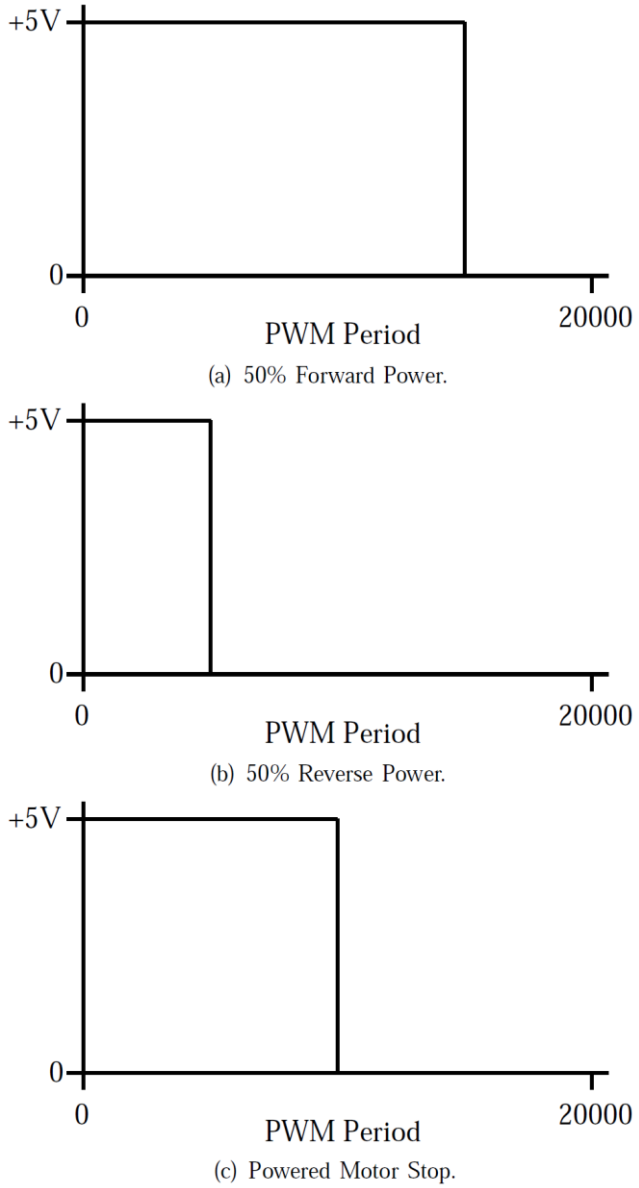

Fig.1 PWM Motor Control with an H–Bridge.

2.3 Motor Control

The motors are controlled using a standard pulse width modulation approach, taking into account that an H–bridge motor driver is used. Two duty cycle registers are utilised, one for each motor, with forward and reverse control. Figure 1 depicts the forward, reverse, and powered stop control of a single motor using PWM through an H–bridge motor driver. Our API provides three functions for controlling the motors:

Fig.2 KITTDASH9 on a Sumo Arena.

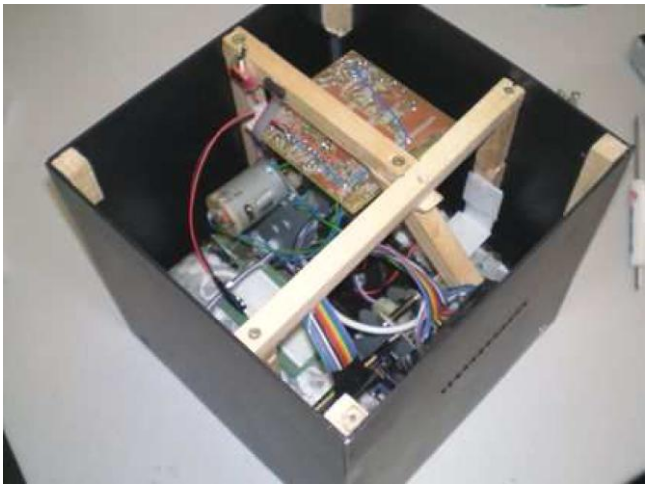

Fig.3 The Interior of KITTDASH9.

```
// Initialise the motor control system
void MotorControlPWM_Init (void );
// Set the motor speed of both motors
void MotorSpeed (int motorLeft ,
int motorRight );
// Turn a choice of motors off
void MotorOff(int choice );
```

The MotorControlPWM_Init() function needs to be called before motor speeds can be controlled. This function sets up the two duty cycle registers and organises the relevant pins for PWM output. The MotorSpeed(left, right) function takes integers as percentage values i.e. calling MotorSpeed(-25, 75) causes the left motor to turn in reverse with 25% power (not speed – generally power to speed is a non-linear relationship) and the right motor to turn forward with 75% power. The MotorOff(choice) function turns off one or more motors when passed one of three constants: MOTORLEFT, MOTORRIGHT or ALLSTOP. If the function is called with MOTORLEFT or MOTORRIGHT then the respective motor is stopped with a powered stop (see Figure 1(c)), if called with ALLSTOP then PWM is switched off (PWM timer base is disabled), switching off power to the motors and letting the motors drift.

3. Application Case Studies

The platform introduced in this paper has been used in a variety of projects including an inverted pendulum robot, balancing weight robot, an autonomous Dr Who Dalek, a sumo fighting robot and an autonomous helicopter. We focus on the latter two for our application case studies of the hardware and software as they are on the opposite ends of the higher education spectrum.

The first case study looks at a robot built by first year students on our Artificial Intelligence and Robotics Bachelors degree. This robot took part in the standard sumo competition at the 2009 Robot Challenge in Vienna. The second case study investigates how a compact version of the same system was used to control an autonomous helicopter for a Masters dissertation and later on in a PhD project.

3.1 Sumo Robot – KITTDASH9

KITTDASH9 was built by a group of first year undergraduate students studying Artificial Intelligence and Robotics at De Montfort University. The students built the robot within the robot club which runs once a week and not during formal teaching time. The robot was designed and built to be entered in the standard class of the robot sumo competition at the Robot Challenge 2009. Figures 2 and 3 show the KITTDASH9 including the mounted embedded system (note that it is mounted upside down) and drive train.

The robot has four custom built light intensity sensors, one on each corner and a modified serial ball mouse to provide a basic form of odometry. The robot has no range finding or bump sensors. Locomotion is provided by two independently driven tracks fitted with a high traction rubber surface. The robot is fitted with a lighting effect system consisting of an array of red LEDs controlled by a separate PIC which is connected to the main embedded system being discussed here.

The students implemented a finite state machine control architecture, as depicted in Figure 4. Each state has a clear control objective which is implemented through a combination of the timer and motor control functions from our API. Transitions between the states are enacted by a combination of states from the light intensity sensors, given on the state transition diagram as a binary string, for example 0101. Notice the light intensity sensors give binary readings. The students achieved this by taking readings from the light intensity sensors, using the ADC part of our API, and putting them through a hard limiter to decide whether the sensor is over a white surface or a black surface – the only two surfaces the robot will encounter during a sumo battle. Each sensor has an individual threshold, allowing each sensor to be individually calibrated.

As mentioned earlier, KITTDASH9 is fitted with a modified serial mouse. Although the students did not manage to use this sensor in their control process, they did (with significant help) manage to get readings from the mouse unit. The mouse was connected directly to the second serial connection on the embedded system. As the mouse ball moves, events are generated and data giving the amount of motion in the *x* and *y* axis are sent on the serial bus. Each event consists of three 7 bit words (see Table II) and the motion reading must be decoded from these three words as given below:

d*x* = word1 & 0x03 << 6 + word2 & 0x3F

d*y* = word1 & 0x0C << 4 + word3 & 0x3F

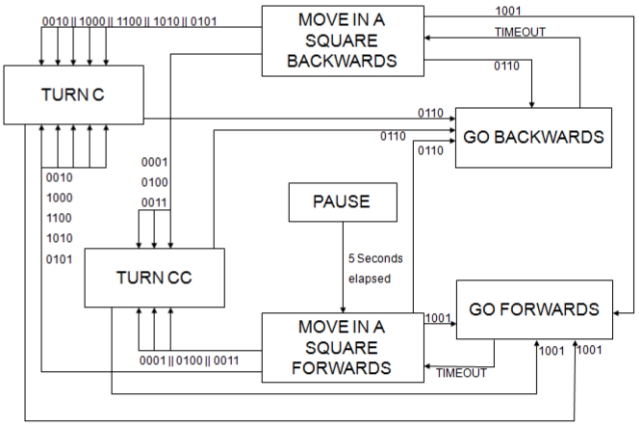

Fig.4 The Finite State Machine Control Architecture as a State Transition Diagram.

	D6	D5	D4	D3	D2	D1	D0
1st Word	1	LB	RB	Y7	Y6	X7	X6
2nd Word	0	X5	X4	X3	X2	X1	X0
3rd Word	0	Y5	Y4	Y3	Y2	Y1	Y0

Tab.2 Microsoft Serial Mouse Protocol [9]

Most of the code to read the serial port was written by the authors, however the students had to decode the readings from the mouse. This meant they got practical experience using bit masking and bit shifting; both of which are taught to students, but rarely covered in practice.

The robot was finished on time and the code written mainly by a group of first year undergraduate students. This would not have been possible without the pre-built embedded system and programming API ready to use. Unfortunately the robot only performed moderately well in competition, it appeared to be under powered compared to its rivals. The high traction rubber meant the robot defended well but it lacked the power to push opposing robots out of the arena.

3.2 Autonomous Helicopter – Flyper

Our proposed hardware and software platform has also been used to create an autonomous helicopter called Flyper. This robot, as shown in Figure 5, has been built by a post graduate for his Master of Science dissertation and later on used in his Doctoral training. The robot's embedded system and software architecture are like the platform design introduced in this paper but the circuitry has been miniaturised to save space and weight.

In general, helicopters have 3 rotational degrees of freedom (DOF), called pitch, roll and yaw, as well as 3 translational DOF called up / down, left / right and forwards / backwards. The helicopter used in this work is a Twister Bell 47 small indoor helicopter model. It is a coaxial rotor helicopter with twin counter rotating rotors with fixed collective pitch and 340 mm span. The rotors are driven by two high performance direct current motors and two servos control the rotor blades' plane angles. The weight of the helicopter in its original state is approximately 210 grams and it can lift up to 120 grams. Before modification, the helicopter was remote controlled by a pilot handling four controls simultaneously: the amount of lift, heading, pitch and roll.

Due to the limited payload the small helicopter is able to carry, the student reduced the platform's physical size by using a prototyping board rather than a PCB. This reduced the size from 80 x 80 mm to 52 x 33 mm and from 51 grams to 25 grams without heat sinks.

In order to keep the autonomous helicopter at a low cost, the student chose to use standard sensors that were already available to him: sonar distance sensors (SRF08)

Fig.5 Autonomous Helicopter Flyper based on our Proposed Platform

for measuring altitude and attitude and a digital compass (CMPS03) to determine the heading. The I^2C bus was used to connect and read the sensors using the PIC microcontroller. Figure 5 shows three sonar sensors mounted on the helicopter as well as the digital compass at the far end of the tail. In order to avoid reflections received by one sonar but transmitted from another, the sensors have been installed at an angle of 10° away from the centre of the helicopter. With this configuration in place and given a flat ground, the attitude of the helicopter can be determined by analysing the difference in measured distances between the sensors. Although the accuracy of the calculated attitude is restricted to the accuracy and resolution of the sonar sensors, the system showed to work as intended.

The PWM outputs together with the L298N motor driver were set to power the two brushed DC motors driving the rotors over a two cogwheel transmission. A small alteration to the circuitry changed the use of the H-bridge as such to using it as a simple driver. This configuration provided the motors with the required power although the motor driver partially reached its peak output current of 4 ampere (e.g. during takeoff).

Within only three months, the student built an autonomous helicopter that achieved relatively stable flight (For test flight videos please visit www.youtube.com/thecci). Furthermore, during his Doctoral training he used this robot to study the use of evolutionary algorithms to tune and optimise conventional proportional integral derivative (PID) control algorithms directly on the robot [10], [11].

4. Conclusions

In this paper we introduced a low cost platform to be used extensively in the broad spectrum of higher education. The platform can be put together by first year students to learn about electronics, bus systems, and digital technologies. The same students can then program the system using a high level C API. Later on, individual students can build new robots using the existing platform and generate complex programs using Assembler and C. Post-graduate students can use the existing robots to study and compare robots, behaviours, and control architectures.

By using industry-standard components and a modular approach, we have developed a low-cost robot-control platform that may be easily reconfigured to suit some of the general computer science and all levels of the robotics teaching curricula: our platform enables students to learn about electronics, embedded systems, communication, bus systems, high and low level programming, robot architectures, and control algorithms, all in individual stages using the same familiar hardware and software.

References

[1] “Roamer official website,” May 2010, http://valiant-technology.com/uk/pages/roamer_home.php.

[2] “Pixie official website,” May 2010, http://www.swallow.co.uk/pixie/pixie1.htm.

[3] “Bee bot official website,” May 2010, http://www.beebot.org.uk.

[4] V. Douglas, “Robots make computer science personal,” *Communications of the ACM*, vol. 49, pp. 12–25, 2006.

[5] “Lego mindstorms on wikipedia,” May 2010, http://en.wikipedia.org/wiki/Lego_Mindstorms.

[6] “Robix rascal official website,” May 2010, http://www.robix.com/.

[7] J. Adams, S. Turner, S. Kaczmarczyk, P. Picton, and P. Demian, “Problem solving and creativity for undergraduate engineers: findings of an action research project involving robots,” in *International Conference on Engineering Education ICEE*, 2008.

[8] “Mobilerobots research robots website,” May 2010, http://mobilerobots.com/ResearchRobots.aspx.

[9] P. Bourke, “Decoding data from the microsoft serial mouse,” April 2003, available at http:// local.wasp.uwa.edu.au/˜pbourke/dataformats/serialmouse/ or on CDROM from http:// local.wasp.uwa.edu.au/˜pbourke/.

[10] B. N. Passow and M. A. Gongora, “Optimising a flying robot: Controller optimisation using a genetic algorithm on a real-world robot,” in *Proceedings of the International Conference on Informatics in Control, Automation and Robotics, ICINCO’08*. Madeira, Portugal: INSTICC Press, May 2008, pp. 151–156.

[11] B. N. Passow, M. A. Gongora, S. Coupland, and A. A. Hopgood, “Realtime evolution of an embedded controller for an autonomous helicopter,” in *Proc. of the IEEE Intl. Congress on Evolutionary Computation (CEC’08)*, Hong Kong, June 2008, pp. 2538–2545.

Benjamin N. Passow

De Montfort University
Centre for Computational Intelligence
Gateway House
Leicester, LE1 9BH
United Kingdom
E-mail: benpassow@dmu.ac.uk

James Wheeler

De Montfort University
Centre for Computational Intelligence
Gateway House
Leicester, LE1 9BH
United Kingdom
E-mail: jw@dmu.ac.uk

Simon Coupland

De Montfort University
Centre for Computational Intelligence
Gateway House
Leicester, LE1 9BH
United Kingdom
E-mail: simonc@dmu.ac.uk

Mario A. Gongora

De Montfort University
Centre for Computational Intelligence
Gateway House
Leicester, LE1 9BH
United Kingdom
E-mail: mgongora@dmu.ac.uk